Anti-Entropy Bandits for Geo-Replicated Consistency

Benjamin Bengfort
University of Maryland
bengfort@cs.umd.edu

Pete Keleher
University of Maryland
keleher@cs.umd.edu

ABSTRACT

Eventual consistency systems can be made more consistent by improving the visibility of a write, that is the time until a write is fully replicated. Gossip based anti-entropy methods scale well but random selection of anti-entropy partners results in less than efficient replication. We propose a simple improvement to pairwise, bilateral anti-entropy; instead of uniform random selection we introduce reinforcement learning mechanisms which assign selection probabilities to replicas most likely to have information. The result is efficient replication, faster visibility, and higher consistency, while still providing high availability and partition tolerance.

INTRODUCTION

A distributed system is made highly available when individual servers are allowed to operate independently without coordination that may be prone to failure or high latency. The independent nature of the server's behavior means that it can immediately respond to client requests, but that it does so from a limited, local perspective which may be inconsistent with another server's response. If individual servers in a system were allowed to remain wholly independent, individual requests from clients to different servers would create a lack of order or predictability, a gradual decline into inconsistency, e.g. the system would experience *entropy*. To combat the effect of entropy while still remaining highly available, servers engage in *anti-entropy sessions* [10] at a routine interval, a process that occurs in the background of client requests.

Anti-entropy sessions synchronize the state between servers ensuring that, at least briefly, the local state is consistent with a portion of the global state of the system. If all servers engage in anti-entropy sessions, the system is able to make some reasonable guarantees about the timeliness of responses; the most famous of which is that in the absence of requests the system will become consistent, eventually. More specifically, inconsistencies in the form of stale reads can be bound by likelihoods that are informed by the latency of anti-entropy sessions and the size of the system [2]. Said another way, overall consistency is improved in an eventually consistent system by decreasing the likelihood of a stale read, which is tuned by improving the *visibility latency* of a write, the speed at which a write is propagated to a significant portion of servers. This idea has led many system designers to decide that "eventual consistency is consistent enough" [3, 11] particularly in a data center context where visibility latency is far below the rate of client requests, leading to practically strong consistency.

Recently there have been two important changes in considerations for the design of such systems that have led us to reevaluate propagation speed: systems are getting larger and are becoming geographically distributed outside of the datacenter. Scaling an eventually consistent system to dozens or even hundreds of nodes increases the radius of the network, which leads to increased noise during anti-entropy; e.g. the possibility that an anti-entropy session will be between two already synchronized nodes. Geographic distribution and extra-datacenter networks increase the latency of anti-entropy sessions so that inconsistencies become more apparent. Large, geographically distributed systems are becoming the norm – from content delivery systems that span the globe, to mobile applications, to future systems such a automated vehicular networks, and all will require additional consistency guarantees without sacrificing availability.

We propose a new class of adaptive distributed data systems whose replicas monitor their environment and modify their behavior to optimize consistency. Anti-entropy utilizes gossip and rumor spreading to efficiently propagate updates in a deterministic fashion without saturating the network [5, 6, 9]. These protocols utilize uniform random selection of peers to synchronize with, which means that a write occurring at one replica is not efficiently propagated across the network. We propose the use of *multi-armed bandit* algorithms [7, 8] to modify the probability of peer selection in order to optimize for fast, successful synchronizations. The result is a network topology that emerges according to access patterns and network latency, often localizing replicas to produce efficient synchronization, efficiency which lowers visibility latency and increases consistency.

SYSTEM DESCRIPTION

Our system implements an eventually consistent, in-memory key-value store that is totally replicated using anti-entropy [4]; a brief sketch of our implementation follows.

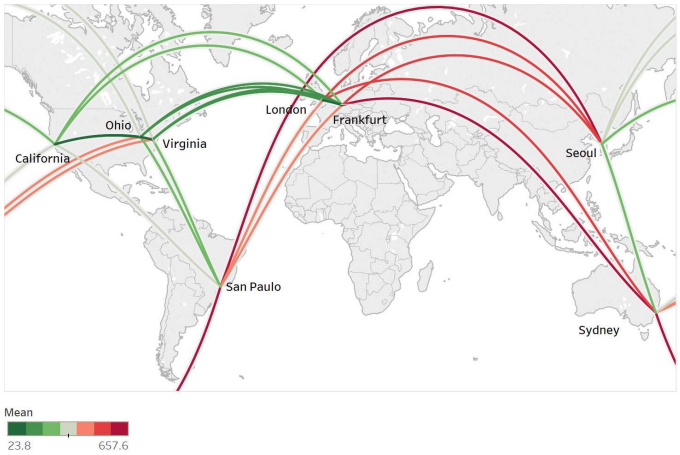

Figure 1: Latencies in a geo-replicated key-value store

PaPoC'18 Worshop Proposal, April 2018, Porto, Portugal

Accesses & Consistency

Clients can `Put` (write) and `Get` (read) key-value pairs to one or more replicas in a single operation. The set of replicas that responds to a client creates a quorum that must agree on the state of the operation at its conclusion. Clients can vary read and write quorum sizes to improve consistency or availability – larger quorums reduce the likelihood of stale reads but smaller quorums respond much more quickly. In large, geo-replicated systems we assume that clients will prefer to choose fewer, local replicas to connect with, optimistic that collisions across the wide-area are rare, e.g. that writes are localized but reads are global.

On `Put`, the instance of the key-value pair is assigned a monotonically increasing, conflict-free *version number* [1]. For simplicity, we assume a fixed number of replicas, therefore each version is made up of two components: the update and precedence ids. Precedence ids are assigned to replicas during configuration, and update ids are incremented to the largest observed value during synchronization. As a result, any two versions generated on `Put` are comparable such that the *latest* version of the key-value pair is the version with the largest update id, and in the case of ties, the largest precedence id.

Additional version metadata including the parent version that object was created from implements a virtual object history that allows us to reason about consistency. Keys can be managed independently, e.g. each key has its own update id sequence, or all objects can be managed together with a single sequence; in the latter case, it is possible to construct an ordering history of operations to all objects. There are two primary inconsistencies that can occur in this system: *stale reads* and *forked writes*. A stale read simply means that the `Get` operation has not returned the globally most recent version of the object. A forked write, on the other hand, means that there are two branches of the object history, created by concurrent writes to the same object. As we will see in the next section, one of these writes will be *stomped* before it can become fully replicated. The ideal consistency for a system is represented by a linear object history without forks.

Anti-Entropy

Anti-Entropy sessions are conducted in a pairwise fashion at a routine interval to ensure that the network is not saturated with synchronization requests. There are two basic forms of synchronization: *push* synchronization is a fire-and-forget form of synchronization where the remote is sent the latest version of all objects, whereas *pull* synchronization requests the latest version of objects and minimizes the size of data transfer. We have implemented *bilateral* synchronization which combines push and pull in a two-phase exchange.

Bilateral anti-entropy starts with the initiating replica sending a vector of the latest local versions of all keys currently stored (this can be optimized with Merkel trees to make comparisons faster). The remote replica compares the versions sent by the initiating replica with its current state and responds with any objects whose version is *later* than the initiating replicas's as well as another version vector of requested objects that are earlier on the remote. The initiating remote then replies with the requested objects, completing the synchronization. We refer to the first stage of requesting later objects from the remote as the pull phase, and the second stage of responding to the remote the push phase.

	Pull	**Push**	**Total**
Synchronize at least 1 object	0.25	0.25	0.50
Synchronize multiple objects	0.05	0.05	0.10
Latency <= 5ms (local)	0.10	0.10	0.20
Latency <= 100ms (regional)	0.10	0.10	0.20
Total	*0.50*	*0.50*	*1.00*

Table 1: Reward Function

There are two important things to note about this form of synchronization. First, this type of synchronization implements a *latest writer wins* policy. This means that it is possible that not all versions become fully replicated – if a later version is written during propagation of an earlier version, then the earlier version gets stomped by the later version. If there are two concurrent writes, only one write will become fully replicated. Second, the visibility latency is maximized when all replicas choose a remote synchronization partner that does not yet have the update. This means that maximal visibility latency is equal to $t \log_3 n$ where t is the anti-entropy interval and n is the number of replicas in the network. However, because of stomps and noise created by uniform random selection of synchronization partners, this latency is never practically achieved.

BANDIT APPROACHES

Multi-armed bandits refer to a statistical optimization procedure that is designed to find the optimal payout of several choices that each have different probabilities of reward and are often used in active and reinforcement machine learning. A bandit problem is designed by identifying several "arms", so called because multi-armed bandit refers to pulling slot machine arms, as well as a reward function that determines how successful the selection of an arm is. During operation, the bandit uses a *strategy* to select an arm – the most common of which is epsilon greedy – then updates the rewards of the selected arm, normalized by the number of selections. The arm with the highest reward value is considered the optimal arm.

Bandits must balance exploration of new arms with possibly better reward values and exploitation of an arm that has higher rewards than the other. In the epsilon greedy strategy, the bandit will select the arm with the best reward with some probability $1 - \epsilon$, otherwise it will select any of the arms with uniform probability. The smaller ϵ is, the more the bandit favors exploitation of known good arms, the larger ϵ is, the more it favors exploration. If $\epsilon = 1$ then the algorithm is simply uniform random selection. A simple extension of this is annealing epsilon greedy, which starts with a large ϵ, then as the number of trials increases, steadily decreases ϵ on a logarithmic scale.

Peer selection for anti-entropy is usually conducted with uniform random selection. To extend anti-entropy with bandits, we design a selection method whose arms are remote peers and whose rewards are determined by the success of synchronization. The goal of adding bandits to anti-entropy is to optimize selection of

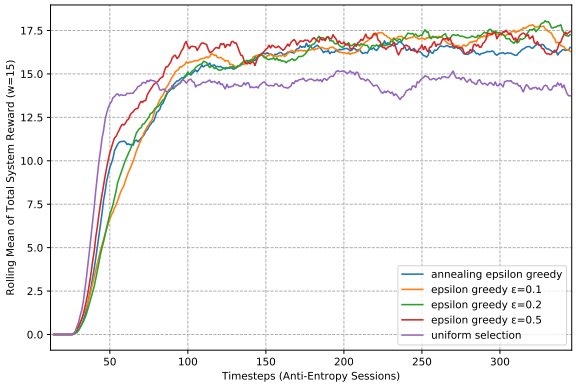

Figure 2: Total system rewards over time

peers such that the visibility latency becomes closer to the optimal propagation time as a synchronization topology emerges from the bandits. A secondary goal is to minimize anti-entropy latency by preferring local (in the same data center) and regional (e.g. on the same continent) connections.

Our reward function favors synchronizations to replicas where the most writes are occurring by giving higher rewards to anti-entropy sessions that exchange later versions in either a push or a pull as well as additional rewards if more than one object is exchanged. Additionally, the latency of the synchronization RPC is computed to reward replicas that are in close proximity to each other. The complete reward function is given in Table 1: for each phase of synchronization (push and pull), compute the reward as the sum of the propositions given. For example if a synchronization results in 3 objects being pulled in 250ms, and 1 object being pushed in 250ms, the reward is 0.75.

EXPERIMENTS

We conducted experiments using a distributed key-value store totally replicated across 24 replicas in 8 geographic regions on 5 continents around the world as shown in Figure 1. Replicas were hosted using AWS EC2 t2.micro instances and were connected to each other via internal VPCs when in the same region, using external connections between regions. The store, called Honu, is implemented in Go 1.9 using gRPC and protocol buffers for RPC requests; all code is open source and available on GitHub.

The workload on the system was generated by 8 clients, one in each region and colocated with one of the replicas. Clients continuously created Put requests for random keys with a unique prefix per-region (consistency conflicts only occur within a region). The average throughput generated per-client was 5620.4 puts/second. Because the mean round trip latency between each region ranged from 23ms to 600ms and synchronization requires at least two round trips, the anti-entropy delay for these experiments was set to 1 second. To account for lag between commands sent to replicas in different regions, the experiment was run for 10 minutes, the workload running for 8 minutes, offset by one minute from the start.

Our first experiments compared uniform random peer selection with epsilon greedy bandits using $\epsilon \in \{0.1, 0.2, 0.5\}$ as well as an annealing greedy epsilon bandit. The total system rewards as a rolling mean over a time window of 15 synchronizations are shown in Figure 2. The rewards ramp up from zero as the clients come online and starting creating work to be synchronized. All of the bandit algorithms improve over the baseline of uniform selection, not only generating more total reward across the system, but also introducing less variability with lower ϵ values. None of the bandit curves immediately produces high rewards as they explore the reward space; lower ϵ values may cause exploitation of incorrect arms, while higher ϵ values take longer to find optimal topologies.

We have visualized the resulting topologies as network diagrams in Figure 3 (uniform selection), Figure 4 (epsilon greedy $\epsilon = 0.1$), Figure 5 (epsilon greedy $\epsilon = 0.2$), and Figure 6 (annealing epsilon). Each network diagram shows each replica as a vertex, colored by region e.g. purple is California, light blue is Sao Paulo, Brazil. Each vertex is also labeled with the 2-character UN country or US state abbreviation as well as the replica's precedence id. The size of the vertex represents the number of `Put` requests that replica received over the course of the experiment; larger vertices represent replicas that were colocated with workload generators. Each edge between vertices represents the total number of successful synchronizations, the darker and thicker the edge is, the more synchronizations occurred between the two replicas. Edges are directed, the source of the edge is the replica that initiated anti-entropy with the target of the edge.

Comparing the resulting networks, it is easy to see that more defined topologies result from the bandit-based approaches. The uniform selection network is simply a hairball of connections with a limited number of synchronizations. Clear optimal connections have emerged with the bandit strategies, dark lines represent extremely successful synchronizations between replicas, while light lines represent connections that happen more rarely. The fewer edges in the graph, the fewer network connections that exist between the links described.

DISCUSSION

CONCLUSION

All code for the key-value store and bandit-based entropy as well as experimental results is open source and available on GitHub at https://github.com/bbengfort/honu.

REFERENCES

[1] Paulo SÃĺrgio Almeida, Carlos Baquero, and Victor Fonte. [n. d.]. Version stamps-decentralized version vectors. In *Distributed Computing Systems, 2002. Proceedings. 22nd International Conference on* (2002). IEEE, 544–551. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1022304

[2] Peter Bailis, Shivaram Venkataraman, Michael J. Franklin, Joseph M. Hellerstein, and Ion Stoica. [n. d.]. Quantifying eventual consistency with PBS. 23, 2 ([n. d.]), 279–302. http://link.springer.com/article/10.1007/s00778-013-0330-1

[3] David Bermbach and Stefan Tai. [n. d.]. Eventual consistency: How soon is eventual? An evaluation of Amazon S3's consistency behavior. In *Proceedings of the 6th Workshop on Middleware for Service Oriented Computing* (2011). ACM, 1. http://dl.acm.org/citation.cfm?id=2093186

[4] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. [n. d.]. Dynamo: amazon's highly available key-value store. In *ACM SIGOPS Operating Systems Review* (2007), Vol. 41. ACM, 205–220. http://dl.acm.org/citation.cfm?id=1294281

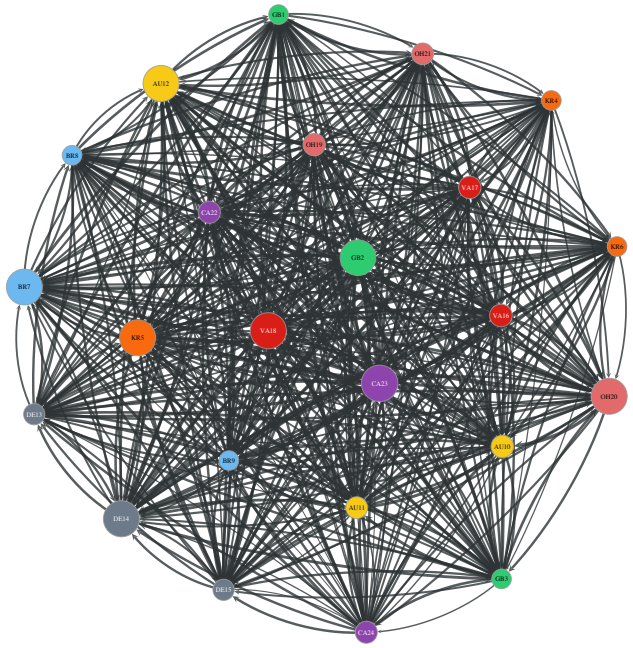

Figure 3: Uniform Selection

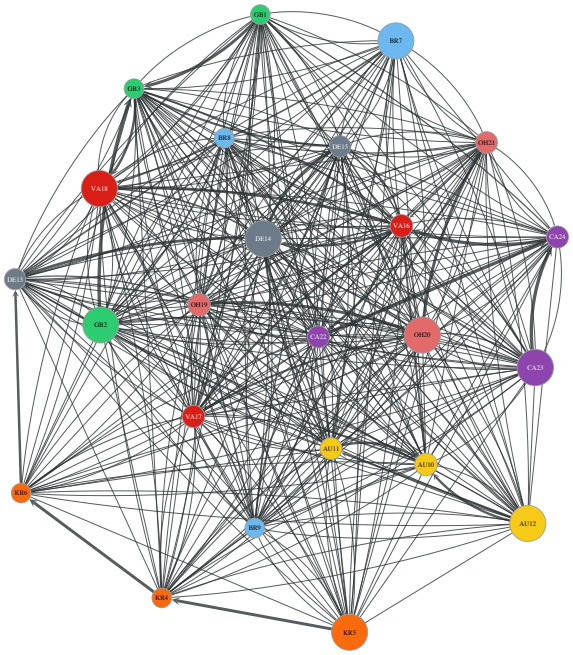

Figure 4: Epsilon Greedy $\epsilon = 0.1$

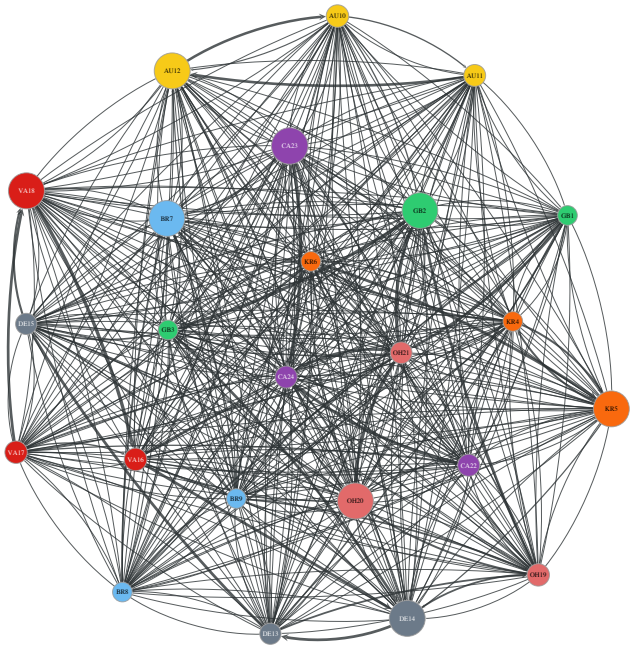

Figure 5: Epsilon Greedy $\epsilon = 0.2$

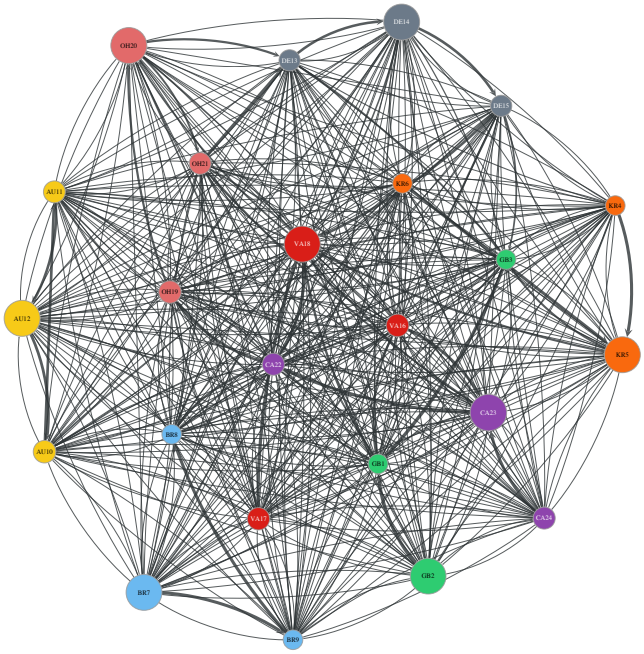

Figure 6: Annealing Epsilon

[5] Bernhard Haeupler. [n. d.]. Simple, fast and deterministic gossip and rumor spreading. 62, 6 ([n. d.]), 47. http://dl.acm.org/citation.cfm?id=2767126

[6] Richard Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vocking. [n. d.]. Randomized rumor spreading. In *Foundations of Computer Science, 2000. Proceedings. 41st Annual Symposium on* (2000). IEEE, 565–574. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=892324

[7] John Langford and Tong Zhang. [n. d.]. The Epoch-Greedy Algorithm for Multi-Armed Bandits with Side Information. In *Advances in Neural Information Processing Systems* (2008). 817–824. bibtex: langford_epoch-greedy_2008.

[8] Haipeng Luo, Alekh Agarwal, and John Langford. [n. d.]. Efficient Contextual Bandits in Non-stationary Worlds. ([n. d.]).
[9] Yamir Moreno, Maziar Nekovee, and Amalio F. Pacheco. [n. d.]. Dynamics of rumor spreading in complex networks. 69, 6 ([n. d.]), 066130. http://journals.aps.org/pre/abstract/10.1103/PhysRevE.69.066130
[10] Douglas B. Terry, Alan J. Demers, Karin Petersen, Mike J. Spreitzer, Marvin M. Theimer, and Brent B. Welch. [n. d.]. Session guarantees for weakly consistent replicated data. In *Parallel and Distributed Information Systems, 1994., Proceedings of the Third International Conference on* (1994). IEEE, 140–149. http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=331722
[11] Hiroshi Wada, Alan Fekete, Liang Zhao, Kevin Lee, and Anna Liu. [n. d.]. Data Consistency Properties and the Trade-offs in Commercial Cloud Storage: the Consumers' Perspective.. In *CIDR* (2011), Vol. 11. 134–143.